Adobe® PostScript®

Matching RGB Color from Monitor to Printer

Adobe Developer Support

Technical Note #5122

14 February 1992

Adobe Systems Incorporated

Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/

EXHIBIT
5

PN LPS5122

Contents

Matching RGB Color from Monitor to Printer

1 Introduction

A source of chronic disappointment for users of color applications is the difference between the colors they see on a display device when they design an image, and the colors they get when they print that image. A related source of disappointment is the difference between colors seen for the same document when reproduced on different color printers.

Although a color specification used on one device can be converted to a second color specification that can be used on a second device to produce the same color, this is not easy when going from monitor to printer. Also, such conversion still does not provide a single document that will reproduce with consistent color across printers.

Applications that use the new color features available in PostScript™ Level 2 can produce color documents that print with *matching color* from a given monitor to any Level 2 color device. There is no need to match colors by converting color specifications for use on a printer. No specific knowledge of the Level 2 printer to be used is required. The color documents created by such applications can be reproduced with matching color on any properly calibrated Level 2 color device.

Note *The term "device" is used here because printers are not the only kind of device that can have PostScript Level 2 interpreters available.*

2 Device-Independent Color and Color Documents

The practice of *characterizing* color devices, and using those characterizations for matching color between particular devices, is not new. What is new with the advent of PostScript Level 2 is the ability it provides to divorce the *production* of color documents from their *reproduction* with matching color.

It is not necessary to specify colors in a selected *standard color space* to make them device-independent (although Level 2 does support various standard color spaces, as well as the RGB spaces discussed here). Instead, the color space where the colors in question happen to be specified can be described in terms of a standard color space, or *reference color space*. The color *specifications* together with the description of their color space are *device-independent*. Device-independent color has the potential to be displayed or printed anywhere as originally intended, subject to the limitations of the devices involved.

Note *Some compromise must be made when the device cannot produce a requested color. A discussion of color gamut, white point adjustment, and other topics relevant to matching color across very different devices is outside the scope of this paper. A wealth of material is available dealing with color reproduction and color science. The "PostScript Language Reference Manual, Second Edition" lists several books in its bibliography that can be consulted for further information.*

PostScript Level 2 devices provide support for device-independent color by allowing applications to describe the color space in which they work using the Level 2 operator **setcolorspace**, and to then use their (now *calibrated*) RGB values unaltered with either **setcolor** or the dictionary form of the **image** operator. The resulting document has device-independent color specifications and will print with matching color on any Level 2 color printer.

Applications which use Level 2 color support to provide matching color from monitor to printer must know the color characteristics of the relevant monitor, and how to describe those characteristics using **setcolorspace**.

3 Monitor Color Characteristics

Color monitors are often characterized in terms of the following:

- *Phosphor chromaticity coordinates* for each red, green, blue phosphor.

  These coordinates provide information about how red the red is, how green the green, and so on. One monitor's red might be bluer than the next monitor's red, for example. This must be taken into account when describing the monitor's color capabilities. Phosphor chromaticities, once determined, are relatively constant over the life of a monitor.

- *White point chromaticity coordinates*.

  These serve as an indication of the color cast of the full white currently available on the display. Unlike phosphor chromaticity, white point chromaticity is readily altered, either intentionally or unintentionally, by users or their software. Users can use a bluish white point when designing images for video applications, and a warmer (redder) white point when designing an image to be reproduced on paper.

- A *gamma function* for each phosphor.

  Gamma functions map the specified RGB values to the intensity levels used for those values when displayed. Each phosphor can have a different gamma function; often all three use the same one. Gamma functions are frequently specified as exponents to be applied to the RGB component values. Like the white point, the gammas in use on a given monitor can be changed.

Table 1 provides an example set of monitor characteristics.

Table 1 *RGB phosphor and white point chromaticity coordinates*

Phosphor	*R*	*G*	*B*	*White point*
x coordinate	0.6250	0.2800	0.1550	0.3127
y coordinate	0.3400	0.5950	0.0700	0.3290
Gammas	1.8	1.8	1.8	

The monitor characteristics listed in Table 1 must be known or approximated to describe the *monitor color space* to a Level 2 interpreter. The more accurate the monitor description, the more closely colors can be matched. There are some additional characteristics which are not required, but which can be used to describe the monitor color space more closely. Default values will be used for such unspecified characteristics; for instance, no *black point* was given above, and so a default black point will be assumed. Such additional information should be used, when available. See the *PostScript Language Reference Manual, Second Edition* for details.

4 Setting the Monitor Color Space

Once the current color characteristics of the display device are known, the appropriate call to **setcolorspace** can be built. For a full description of the **CIEBasedABC** color space and the **setcolorspace** operator, refer to the *PostScript Language Reference Manual, Second Edition*.

The following code establishes a color space corresponding to the monitor characteristics shown in Table 1:

```
[/CIEBasedABC <<
   /DecodeLMN [{1.8 exp} bind dup dup]
   /MatrixLMN [0.4497 0.2446 0.0252
               0.3163 0.6720 0.1412
               0.1845 0.0833 0.9227]
   /WhitePoint [0.9505 1 1.0890]
>>] setcolorspace
```

The **CIEBasedABC** color space must be selected if matching color is desired. The entries shown in the dictionary are constructed from the monitor's known characteristics. Equations that produce the values needed for the **MatrixLMN** and **WhitePoint** entries from the available chromaticity information are given below.

$$z = y_W \times ((x_G - x_B) \times y_R - (x_R - x_B) \times y_G + (x_R - x_G) \times y_B)$$

$$Y_L = \frac{y_R}{R} \times \frac{(x_G - x_B) \times y_W - (x_W - x_B) \times y_G + (x_W - x_G) \times y_B}{z}$$

$$X_L = Y_L \times \frac{x_R}{y_R} \qquad Z_L = Y_L \times \left(\frac{1 - x_R}{y_R} - 1\right)$$

$$Y_M = -\frac{y_G}{G} \times \frac{(x_R - x_B) \times y_W - (x_W - x_B) \times y_R + (x_W - x_R) \times y_B}{z}$$

$$X_M = Y_M \times \frac{x_G}{y_G} \qquad Z_M = Y_M \times \left(\frac{1 - x_G}{y_G} - 1\right)$$

$$Y_N = \frac{y_B}{B} \times \frac{(x_R - x_G) \times y_W - (x_W - x_G) \times y_R + (x_W - x_R) \times y_G}{z}$$

$$X_N = Y_N \times \frac{x_B}{y_B} \qquad Z_N = Y_N \times \left(\frac{1 - x_B}{y_B} - 1\right)$$

$$\begin{aligned} X_W &= R \times X_L + G \times X_M + B \times X_N \\ Y_W &= R \times Y_L + G \times Y_M + B \times Y_N \\ Z_W &= R \times Z_L + G \times Z_M + B \times Z_N \end{aligned}$$

Table 2 shows the correspondence between monitor characteristic values and the notation used in the equations.

Table 2 *Monitor characteristics and corresponding variable names*

Phosphor	*R*	*G*	*B*	*White point*
x coordinate	x_R	x_G	x_B	x_W
y coordinate	y_R	y_G	y_B	y_W
Gammas	gamma_R	gamma_G	gamma_B	

Note chromaticity coordinates (lower case) are being used to get CIE XYZ tristimulus values (upper case). When gamma functions are given in exponent form, they can be used with calls to the PostScript language operator **exp** to build the decode functions.

Once the tristimulus values have been determined, they are used as follows:

```
[/CIEBasedABC <<
   /DecodeLMN [ {gamma_R exp} bind
                {gamma_G exp} bind
                {gamma_B exp} bind]
   /MatrixLMN [X_L Y_L Z_L
                X_M Y_M Z_M
                X_N Y_N Z_N]
   /WhitePoint [X_W Y_W Z_W]
>>] setcolorspace
```

The code in Example 1 demonstrates building the dictionary part of the call to **setcolorspace** from monitor chromaticity coordinates and gamma exponents:

Example 1: *Monitor characteristics to* ***setcolorspace*** *dictionary*

```
/*
* Utility function returning a single set of tristimulus values
*/

static void makeXYZ (double sign, double z, double x1, double y1,
             double x2, double y2, double x3, double y3,
             double xW, double yW, double *X, double *Y, double *Z)
{
  *Y = sign * y1 * ((yW * (x2 - x3) - y2 * (xW - x3) + y3 * (xW - x2))
             / z);
  *X = *Y * (x1 / y1);
  *Z = *Y * (((1 - x1) / y1) - 1);
}  /*  end function makeXYZ  */

/*
* Builds a string creating a dictionary for setcolorspace based
* on monitor chromaticity coordinates and gamma exponent values.
* Returns a pointer to that string, if successful.
*/

char *makeCIEABCdict (double xR, double yR, double xG, double yG,
                double xB, double yB, double xW, double yW,
                double gR, double gG, double gB)
{
  char *stringPS;
  int numchars;
  double z, XL, YL, ZL, XM, YM, ZM, XN, YN, ZN, XW, YW, ZW;

  getMem (stringPS, 256, char, errorExit ("makeCIEABCdict:
                                  \malloc failed\n"););

  z = yW * (yR * (xG - xB) - yG * (xR - xB) + yB * (xR - xG));

  makeXYZ (1, z, xR, yR, xG, yG, xB, yB, xW, yW, &XL, &YL, &ZL);
  makeXYZ (-1, z, xG, yG, xR, yR, xB, yB, xW, yW, &XM, &YM, &ZM);
  makeXYZ (1, z, xB, yB, xR, yR, xG, yG, xW, yW, &XN, &YN, &ZN);

  XW = XL + XM + XN; YW = 1; ZW = ZL + ZM + ZN;

  if ((numchars = sprintf (stringPS, "3 dict 3 {dup} repeat\n\
/DecodeLMN [{\n\
%.2f exp} bind {\n%.2f exp} bind {\n%.2f exp} bind] put\n\
/MatrixLMN [\n %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f ] put\n\
/WhitePoint [\n%.4f %.4f %.4f ] put\n", gR, gG, gB,
XL, YL, ZL, XM, YM, ZM, XN, YN, ZN, XW, YW, ZW)) < 100)
    errorExit ("makeCIEABCdict: couldn't build string\n");

  return stringPS;

}  /*  end function makeCIEABCdict  */
```

Although the way the monitor characteristics are expressed above is the most common, this same information can be given in another form. The chromaticity information used above can still be derived from these alternate forms. As noted earlier, there are many good references on these topics.

5 Setting the Color Values

Once the color space has been described and set with **setcolorspace**, the color data can be sent and will be reproduced as it appeared in the color space just described.

Note *This occurs to the extent possible. There are colors monitors are capable of displaying that no printer can reproduce, and vice versa. Ways of dealing with this are not discussed in this paper.*

Applications developers should be aware that although the RGB values do not have to be altered, they must be used with **setcolor** or the dictionary form of the **image** operator to have those values interpreted according to the color space just set. As documented in the *PostScript Language Reference Manual, Second Edition*, colors set with device-dependent color operators such as **setrgbcolor** will not be converted, and the color space will be set to the corresponding device-dependent one.

Note *If you want to use a device-dependent color operator after having set a CIE-based color space, that section of code can be bracketed with **gsave**/ **grestore** to avoid altering the global color state.*

6 Level 2 Printers and Device-Independent Color

Previous sections have detailed the production of a PostScript Level 2 document with device-independent calibrated RGB color specifications. Strictly speaking, using the operators **setcolorspace**, **setcolor**, and the dictionary form of the **image** operator correctly are all that an application producing a document with portable color need care about. On the other hand, it should be apparent that there is a lot of work going on in the Level 2 printer to provide the color matching and readers may be curious about what that work is.

Conceptually speaking, two things happen on a Level 2 printer when a CIE-based color space is used:

- The interpreter uses the color space description provided by the application to map the colors specified to their values in the reference color space. This is a device-space to standard-space conversion (monitor to standard).
- The interpreter then takes these reference color space values and turns them into corresponding printer colors as directed by the current color rendering dictionary. A default color rendering dictionary has been developed for each Level 2 color printer to provide matching color across printer models and instances. This is a standard-space to device-space conversion (standard to printer).

In practice, these tasks may not be done as separate steps. The PostScript interpreter combines multiple operations into a single compound operation whenever possible, both to improve performance and to avoid a loss of numerical precision.

The behavior of the default color rendering dictionary is to provide matching color when possible, for the usual paper stocks, inks, and so on. Color devices, which can be used with different paper stocks, different inks, and so on, may provide a variety of color rendering dictionaries, to be chosen at print time according to the media used.

Like the availability of alternate paper trays, the availability of alternate color rendering dictionaries for a particular printer is *device-dependent*. Color rendering dictionary selection, if some dictionary other than the default is desired, should be done at print time, as part of printer setup, and, like paper tray selection, should not be done as part of any PostScript language document intended to be portable.

Any invocation of **setcolorrendering** within a PostScript language document defeats the division between the device-independent color specification and the device-dependent color conversion that allows these documents to be color portable.

7 Device-Independent Color and Level 1 Devices

PostScript Level 2 documents that are device-independent with respect to color will not be printed with matching color on Level 1 devices. The machinery supporting matching color on Level 2 devices is not present on Level 1 devices.

Even though the support for matching color is not present, PostScript Level 2 documents should not fail to print on Level 1 devices. As for many Level 2 features, self configuring code can be used in the documents that will use the Level 2 features where available, and do something reasonable where not.

Several papers have been written about emulating Level 2 features and creating self configuring code. See

- the *PostScript Language Reference Manual, Second Edition*, Appendix D, *Compatibility Strategies*
- Technical Notes #5111, “Emulation of the **setstrokeadjust** Operator;” #5112, “Emulation of the **makepattern** and **setpattern** Operators;” #5113, “Emulation of the **execform** Operator;” and #5123, “Emulation of the **rectclip**, **rectfill**, and **rectstroke** Operators.”

Some of the emulations described in the papers are not complete, in that not all aspects of the Level 2 features are readily emulated on Level 1. Because the CIE-based color spaces are not available on Level 1, emulations of **setcolorspace**, **setcolor**, and the dictionary form of the **image** operator fall into the class for which only partial emulations can be constructed.

One such partial emulation of **setcolor** and **setcolorspace** is shown below. All the device color spaces that can be set with **setcolorspace** can of course be set using these emulations. As a bonus for Level 1 systems, the emulations include one for the **setcmykcolor** operator, which is not present on all Level 1 systems. The **Indexed** color space (in its device-dependent incarnations) is also provided here.

The CIE-based color spaces will be mapped to the device color space that has the corresponding number of color components; if **CIEBasedABC** is selected, the emulation will use **setrgbcolor** for the three color components. For calibrated RGB color spaces, this will at least leave the appearance of the document no worse off than if **setrgbcolor** had been used to begin with. An application using this code as is would have to limit itself to setting calibrated RGB color spaces. It would also have to note that images are not supported.

The following code is intended as a demonstration of how Level 2/Level 1 compatibility might be achieved. Any serious emulation code for these operators would also include some provision for the dictionary form of the **image** operator.

The emulation package defines two procedures.The scs procedure emulates the Level 2 **setcolorspace** operator, and the sc procedure emulates **setcolor**. This code is self-configuring. If executed on a Level 1 printer, the emulation procedures are used. If executed on a Level 2 printer, **setcolor** is loaded into sc and **setcolorspace** is loaded into scs.

Example 2: *Partial emulations of the **setcolorspace** and **setcolor** operators*

```
% some utility procedures
/bd { bind def } bind def
/ld { load def } bd
/xd { exch def } bd
/gs /gsave ld
/gr /grestore ld

% a place to save color space information
% and procedures for saving and accessing that info
/ColorDict 10 dict def
/pd { ColorDict 3 1 roll put } bd
/gd { ColorDict exch get } bd

% set a boolean indicating language level
/Level2? /languagelevel where {
   pop languagelevel 2 ge { true } { false } ifelse
} { false } ifelse def
```

```
% emulate setcmykcolor if not present, using method shown in
% PLRM Section 6.2.4: Conversion from DeviceCMYK to DeviceRGB
/setcmykcolor where { pop } {
   /setcmykcolor {
      1 sub 4 1 roll
      3 {
         3 index add neg dup 0 lt { pop 0 } if 3 1 roll
      } repeat
      setrgbcolor pop
   } bind def
} ifelse

% the actual setcolorspace and setcolor emulations
Level2? {
   % A level 2 device
   % Define sc and scs to be setcolor and setcolorspace
   pop
   /scs /setcolorspace ld
   /sc /setcolor ld
} {
   % A Level 1 device!
   % Save the number of arguments that each color space takes
   % in the ColorDict dictionary
   /ColorValues 5 dict begin
      /DeviceGray 1 def
      /DeviceRGB 3 def
      /DeviceCMYK 4 def
      /CIEBasedABC 3 def
      /CIEBasedA 1 def
      currentdict
   end pd

   /ColorValue? { /ColorValues gd exch get } bd

   % Define a ColorProc dictionary in the ColorDict.
   % Assign a Level 1 operator to each device color space.
   /ColorProcs 5 dict begin
      /DeviceGray /setgray ld
      /DeviceRGB /setrgbcolor ld
      /DeviceCMYK /setcmykcolor ld
      /CIEBasedABC /setrgbcolor ld
      /CIEBasedA /setgray ld
      currentdict
   end pd

   /ColorProc? { /ColorProcs gd exch get } bd
   % Define setcolorspace partial emulation
   /scs {
      /SetColor {(Unsupported colorspace!) == } pd
      dup /ColorSpace exch pd
      dup type /arraytype eq { 0 get } if    dup dup dup
      % If color space on stack is DeviceRGB, DeviceGray, DeviceCMYK
      % or CIE based, set SetColor component of the ColorDict to an
      % appropriate Level 1 color operator.
      /ColorProcs gd exch known {
         dup ColorProc? /SetColor exch pd
         dup ColorValue? /NumColorValues exch pd
      } if

      % An Indexed color space
      dup /Indexed eq {
      % The Indexed color space provides a color table that allows
      % the PS programs to access arbitrary colors in another space.
      % The operand to setcolorspace (scs) is a 4 element array.
      % These elements are:
      % Indexed: Color space name
      % base: an array or name that identifies the base color space.
      % hival: Max valid index value.
      % lookup: procedure or a string. It provides mapping between
      % the values of index and the colors in the base color space.

         /NumColorValues /ColorSpace gd 1 get ColorValue? pd
         /ColorSpace gd 3 get type /stringtype ne {
            % lookup parameter is procedure!
            /SetColor [
            /ColorSpace gd 3 get
            /exec load
            /ColorSpace gd 1 get ColorProc?
         ] cvx bind pd
      } {
         % lookup parameter is string
         /SetColor [
         /ColorSpace gd 3 get
         /NumColorValues gd
         3 -1 /roll load /mul load
         0 1 /NumColorValues gd 1 sub {
            1 index add
            2 index exch get 255 div
            3 1 roll
         } /for load
         /pop load /pop load
               /ColorSpace gd 1 get ColorProc?
            ] cvx bind pd
         } ifelse % arraytype
      } if
      pop
   } bd

   /sc {
      /SetColor gd exec
   } bd
} ifelse
```

The following examples demonstrate sc and scs usage. Notice that the CIE-based example builds the dictionary part of its operand in a way compatible with Level 1 interpreters.

```
/DeviceRGB scs
.2 .4 .5 sc

[/CIEBasedABC 3 dict 3 {dup} repeat
   /DecodeLMN [{1.8 exp} bind {1.8 exp} bind {1.8 exp} bind] put
   /MatrixLMN [[0.4497 0.2446 0.0252
               0.3163 0.6720 0.1412
               0.1845 0.0833 0.9227] put
   /WhitePoint [0.9505 1 1.0890] put
] scs
.5 .3 .1 sc

[/Indexed /DeviceRGB 255 <000000 FF0000 00FF00 0000FF
B54367 ...> ]scs
2 sc
```

Appendix: Changes Since Earlier Versions

Changes since August 13, 1991

- Rewrote discussion to address application concerns more directly, and to give a step by step description of the actions required to support matching color.
- Changed the title from “Supporting the Calibrated RGB Color Space” to “Matching RGB Color from Monitor to Printer” to better describe the topic of this document.
- Used a different set of example monitor characteristics.
- Described the production of documents with device independent color specifications.
- Emphasized the device specific nature of color rendering dictionaries.
- Provided a C code example converting monitor characteristics to the appropriate dictionary to supply **setcolorspace**.
- Altered the **setcolor**/**setcolorspace** partial emulation.

Index